DROOLS Expert
Lab

Davide Sottara
dsotty AT gmail.com

May 29, 2011

Outline

Outline

Installazione

- Plugin Eclipse :
  http://downloads.jboss.com/drools/updatesite3.5/
- Drools Expert Capitolo 7
  http://downloads.jboss.com/drools/docs

Getting Started

- Creare un Drools Runtime

Eclipse - Preferences - Drools - Installed Drools Runtime

- Creare un nuovo progetto Drools "Hello World"
  - Un file di regole (Sample.drl)
  - Un main Java (DroolsTest.java)

Il codice / Main

```
public static final void main(String[] args) {
  // carica e compila le regole in una RETE
  KnowledgeBase kbase = readKnowledgeBase();
  // (re)inizializzazione della RETE
  StatefulKnowledgeSession ksession =
      kbase.newStatefulKnowledgeSession();

    // inserisci fatti (match e attivazione)
    ksession.insert(...);
    // esecuzione
    ksession.fireAllRules();
        // itera ...

  // oppure chiusura della sessione
  ksession.dispose();
}
```

Il codice / Caricamento

```
private static KnowledgeBase readKnowledgeBase() {
  // creo il "compilatore"
  KnowledgeBuilder kbuilder =
    KnowledgeBuilderFactory.newKnowledgeBuilder();
  // compilo una o piu "risorse"
  kbuilder.add(
    ResourceFactory.newClassPathResource("Sample.drl"),
    ResourceType.DRL);
  // inizializzo la RETE "vuota" e le strutture
  KnowledgeBase kbase =
    KnowledgeBaseFactory.newKnowledgeBase();
  // aggiungo le regole compilate
  kbase.addKnowledgePackages(
      kbuilder.getKnowledgePackages());
  return kbase;
}
```

Let's Run!

Usiamo le *View*

- Capitolo 7.6 della documentazione di Drools Expert
- Impostare un breakpoint su `fireAllRules`
- Passare alla *Drools Perspective*
- Debug!
- Selezionare una delle view (audit, WM, agenda...)
- Selezionare la sessione tra le variabili di debug

Outline

Outline

Un po' di fatti

- Creare una classe bean[1] di tipo `com.sample.Person`
  - `age : int`
  - `name : String`
  - ...
- Creare una classe bean[2] di tipo `com.sample.Mail`
  - `sender : Person`
  - `destination : Person`
  - `body : Message`
- `DroolsTest.java`: creare un po' di messaggi, persone e mail; `insert` ciascuno nella sessione
- Creare un nuovo file di regole e cambiare il file caricato nel main java

[1] campi, costruttore, *getter*, *setter*, `equals`, `hashcode` e `toString` definiti sui field.

[2] Hint: Eclipse consente di generare automaticamente tutti i metodi di cui sopra

Struttura delle regole

Una o piú regole:

```
rule "ID della regola"
when
  // qui i pattern della premessa
then
  // qui le conseguenze
end
```

Inserimenti : verifica

```
rule "Msg"
  when $m : Message()
  then System.out.println($m);
end
rule "Person"
  when $p : Person()
  then System.out.println($p);
end
rule "Mail"
  when $m : Mail()
  then System.out.println($m);
end
```

Le regole scattano per ogni fatto del tipo adeguato

Esercizio

Aggiungere vincoli ad un pattern - per quali oggetti/fatti scattano le regole?

Pattern con filtri (esempio)

```
rule "Person_filter"
  // uso di (), && e || per espressioni
  when $p : Person( name == "john" ,
                    age < 10 ||
                    ( age > 18 && age < 35 )
                  )
  then System.out.println($p);
end
```

Outline

Esercizio : Join

Scrivere una regola che scatta per ogni possibile coppia di Persone:

- una persona fa "coppia" con se stessa?

Esercizi liberi (proposte:)

- trovare le coppie di omonimi
- trovare le coppie formate da un uomo e una donna[3] in cui lui é piú giovane
- e se, avendo gi'a P(X,Y), non volessimo P(Y,X) ??[4][5]

[3] estendere il bean...
[4] con le persone non ha senso, ma ad esempio con le squadre di calcio?
[5] Hint: vedi oltre...

Join

```
rule "Join"
  // hint: binding con "$X :"
  when
    $p1 : Person()
    $p2 : Person()
  then
    System.out.println("R1 :" + $p1 + " vs. " + $p2);
end
rule "Pairs"
  when
    $p1 : Person()
    $p2 : Person( this != $p1 )
  then
    System.err.println("R2 :" + $p1 + " vs. " + $p2);
end
```

Esercizio : Join multipli

"Stampare il testo del messaggio nel body di una Mail[6], a patto che il mittente sia una persona di almeno 25 anni di nome John che ha mandato la mail a se stesso"

[6] leggasi : di ogni mail che soddisfa i vincoli

Join

```
rule "Ex1 (versione 1 - full)"
when
  $p : Person( age >= 25, name == "john" )
  $p2 : Person( this == $p ) // serve davvero?
  $m : Mail( sender == $p,
             destination == $p2,
             $body : body )
  Message( this == $body, $text : message)
then
    System.out.println(
      "Il contenuto del messaggio: " + $text);
end
```

“Dot notation”

```
// X( y.z ... ) => X.getY().getZ() ...
rule "Ex1 (versione 2 - compact)"
when
  Message( $p : sender,
           sender.name == "john",
           sender.age >= 25,
           receiver == $p,
           $text : body.message )
then
    System.out.println(
      "Il contenuto del messaggio: " + $text);
end
```

Outline

Quantificatori

Drools supporta i quantificatori $\exists$ e $\forall$

- **exists** P(...)
  - Nella WM ci deve essere almeno un oggetto che fa match con P(...)

Quantificatori

Drools supporta i quantificatori $\exists$ e $\forall$

- **exists** P(...)
  - Nella WM ci deve essere almeno un oggetto che fa match con P(...)
- **not** P(...)
  - Scatta quando nella WM non c'é nessun oggetto che fa match con P(...)

Quantificatori

Drools supporta i quantificatori $\exists$ e $\forall$

- **exists** P(...)
  - Nella WM ci deve essere almeno un oggetto che fa match con P(...)
- **not** P(...)
  - Scatta quando nella WM non c'é nessun oggetto che fa match con P(...)
- **forall** P(...)
  - Scatta quando tutti gli oggetti di tipo P fanno match con P(...)

Quantificatori

Drools supporta i quantificatori $\exists$ e $\forall$

- **exists** P(...)
  - Nella WM ci deve essere almeno un oggetto che fa match con P(...)
- **not** P(...)
  - Scatta quando nella WM non c'é nessun oggetto che fa match con P(...)
- **forall** P(...)
  - Scatta quando tutti gli oggetti di tipo P fanno match con P(...)
- **forall** ($p : P(...) Q(... $p ...))
  - Scatta quando tutti gli oggetti che fanno match con P(...) fanno match anche con Q(... p ...)

Esercizio : Quantificatori

- Stampare il nome delle persone che hanno ricevuto almeno una mail
- Stampare il nome delle persone che NON hanno ricevuto nessuna mail
- Stampare il nome delle persone che hanno ricevuto tutte (e sole) mail da un mittente di nome "john"

Soluzione / 1

```
rule "Quantifier 1"
when
    $p : Person( $n : name )
    exists Mail( receiver == $p )
then
    System.out.println(
      $n + " ha ricevuto una mail" );
end
```

Attenzione : la regola scatta 0 o 1 volta per persona!

Soluzione / 2

```
rule "Quantifier 2"
when
    $p : Person( $n : name )
    not Mail( receiver == $p )
then
     System.out.println(
       $n + " non ha ricevuto mail" );
end
```

Attenzione : la regola scatta 0 o 1 volta per persona!

Soluzione / 3

```
rule "Quantifier 3"
when
    $p : Person( $n : name )
    forall ( Mail( receiver == $p,
                   $s : sender )
             Person( this == $s, name == "john")
           )

then
     System.out.println(
       $n + " ha ricevuto solo mail da john" );
end
```

Attenzione : la regola scatta 0 o 1 volta per persona!

Outline

Bean "on the fly"

Drools supporta - in modo limitato - la generazione dinamica di classi

```
// nel file .drl
// tipicamente appena prima delle regole
declare Pair
   first : Person
   second : Person
end
```

Ora possiamo usare istanze di `Pair` come bean nelle regole![7]

[7] e anche nel main, con un po' di API apposite. Consultare la documentazione o chiedere per mail...

Insert

- Oltre ad eseguire codice generico, il "**then**" di una regola puó generare nuovi fatti e/o inserirli nella WM

Insert

- Oltre ad eseguire codice generico, il "**then**" di una regola puó generare nuovi fatti e/o inserirli nella WM
- Questi, a loro volta, possono attivare regole in cascata

Chaining

```
rule "Chainer"
when
  $p1 : Person()
  $p2 : Person()
then
  Pair p = new Pair();    // no constructor :(
    p.setFirst($p1);
    p.setSecond($p2);
  insert(p);  // inserisco un nuovo fatto
end

rule "Chained"
when
  $p : Pair()
then
  System.out.println("Generata coppia " + $p);
end
```

Esercizio

"Generare tutte le coppie di Persone ordinate distinte"[8]

[8] vedi l'esercizio proposto precedentemente...

Soluzione

```
rule "Join distinct"
when  // tutte le coppie...
  $p1 : Person()
  $p2 : Person( this != $p1 )
  // != $p1 -> ...di persone diverse...
  not Pair( first == $p2, second == $p1 )
      // "... se non esiste gia' la simmetrica"
then
  Pair p = new Pair();
    p.setFirst($p1);
    p.setSecond($p2);
  insert(p);
end
```

Outline

“From”

Estrae oggetti da una `Collection` anche se non sono nella WM

```
rule "From"
when
   // inserita come List<? extends Person>
   $l : List()
   // join come se fossero nella WM
   Person( ... ) from $l
then
   ...
end
```

Scatta (fino a) `$l.size()` volte

From - esercizio

- Supponendo che una `Person` abbia un campo `children` di tipo `Collection<? extends Person>`
- Scrivere una regola che scatta per ogni figlio di una persona

Soluzione

```
rule "I figli"
when
  $parent : Person( $childz : children )
  $child  : Person( ) from $childz
then
  System.out.println(
    $parent + " genitore di " + $child );
end
```

From - esercizio

- Scrivere il (Print)Visitor di un albero
  - tutti i nodi devono essere attraversati
  - l'ordine non importa (per ora)

Soluzione

```
rule "Print"
when // Node e' una classe opportuna...
  $n : Node()
then
  System.out.println("Visito " + $n);
end

rule "Visitor"
when
  // nel main : insert(root); - e basta
  Node( $childz : children )
  $child  : Node( ) from $childz
then
  insert($child);
end
```

Salience

Che succede quando un fatto attiva piú regole allo stesso tempo?

- Nell'esempio di prima, un `Node` causa ...

Salience

Che succede quando un fatto attiva piú regole allo stesso tempo?

- Nell'esempio di prima, un `Node` causa ...
- 1 + `children.size()` attivazioni!

Salience

Che succede quando un fatto attiva piú regole allo stesso tempo?

- Nell'esempio di prima, un `Node` causa ...
- 1 + `children.size()` attivazioni!

Possibile soluzione:

- Salience : regole con salience piú alta scattano per prime

Salience - esempio

```
rule "R1"
salience 10 // prima questa
when
  $p : Person( age > 18 )
then
  ...
end

rule "R2"
salience 5 // poi questa
when
  $p : Person( age < 60 )
then
  ...
end
```

Il valore assoluto della salience non importa, solo quello relativo

From - esercizio parte II

- Fissare una *salience* per le due regole del Visitor
- Creare il getter `childrenReversed` che restituisce la lista dei figli in ordine opposto a `children`
- Che succede cambiando la salience relativa e/o l'ordine dei figli?

Outline

Collect

Il duale di `from`:

```
rule "Collect"
when
  ...
  $l : List() from collect ( P(...) )
  ...
then
  fai qualcosa con $l
end
```

Estrae tutti gli oggetti che fanno match con P(...) e crea una `List`

Accumulate /1

Operazioni generiche su collezioni di oggetti:

```
// estrai tutti gli oggetti che fanno match con P(...)
from accumulate (
  P(...)
)
```

Accumulate /2

```
from accumulate (
   $p : P(...)
   action ( f($p) )
   // esegui operazioni su ciascuno
)
```

Accumulate /3

```
from accumulate (
   $p : P(...)
   // inizializza le strutture dati necessarie
   init ( x = new X(); )
   action ( x = f(x,$p) )
   // ''accumula'' i risultati parziali
)
```

Accumulate /3

```
Y(...) from accumulate (
   $p : P(...)
   init ( x = new X(); )
   action ( x = f(x,$p) )
   return ( new Y(x) )
   // restituisce un oggetto
   // filtrabile dal pattern Y(...)
)
```

Accumulate - Esempio

```
rule "Accumulate"
when
  // per ogni persona...
  Person( $a : age, $childz : children )
  // ... piu' giovane della...
  Number( doubleValue > $a )
  // ... somma delle eta dei figli :
       from accumulate(
         // estraggo tutti i figli (from!)
         Person( $age : age ) from $childz,
         // init - eseguito una volta all'inizio
         init( double total = 0; ),
         // action - eseguito per ogni figlio
         action( total += $age; ),
         // result - eseguito solo alla fine
         result( new Double( total ) ) )
then ... end
```

Outline

Query

- Consentono di estrarre oggetti dalla Working Memory
- Sfruttano le capacitÃ di filtraggio della RETE
- Possono essere parametriche
- Le variabili dichiarate consentono di accedere ai risultati

Esercizi:

- Estrarre le persone maggiorenni
- Estrarre il nome delle persone di eta' data

Query

```
query "peopleOfAge"
  $p : Person( age > 18 )
end

query "peopleByAge" (int $age )
  Person( $name : name, age == $age )
end
```

Outline

TMS

- Inserimento condizionale
- I fatti vengono ritratti automaticamente quando le premesse non sono piu' valide

Esercizio bonus: Scrivere un semplice servizio pubblicitario che adatti il tipo di reclame all'etÃ delle persone

Ad / 1

```
declare Ad
   target  : String
   message : String
end

query "advertisement" (String $name)
   $adv : Ad( target == $name, $msg : message )
end
```

Ad / 2

```
rule "Kids"
  when
    $p : Person( $name : name, age < 18 )
  then
    insertLogical( new Ad($name,
      "Compra una macchinina giocattolo") );
  end

rule "Men"
  when
    $p : Person( $name : name, age >= 18 )
  then
    insertLogical( new Ad($name,
      "Compra un'automobile sportiva") );
  end
```

Ad / 3

```
declare Birthday
  person : Person
end

rule "Birthday"
dialect "mvel"
when
  $p : Person( $age : age )
  $b : Birthday( person == $p )
then
  System.out.println("Compleanno!");
  // al termine della modify, l'oggetto $p viene ritra
  modify ($p) {
    age = $age + 1;
  }
  retract($b);
end
```

Ad - Java / 5

```
//Intanto, lato Java...
// Ottengo il tipo dichiarato a runtime
FactType birthdayClass = kSession.getKnowledgeBase()
  .getFactType("it.unibo.deis.lia", "Birthday");

// Istanzio e setto gli oggetti
Object bday = birthdayClass.newInstance();
birthdayClass.set(bday, "person", p1);
```